Choose Your Own Project: COVID-19

Kevin Stroop

2024-04-26

Contents

Introduction

This report analyzes the spread of COVID-19 in the United States using county-level data from the first seven months of the pandemic. The goal is to understand the progression of confirmed cases and to forecast future trends using machine learning models. The COVID-19 dataset used in this project includes daily counts of confirmed cases and deaths [1]. This will serve as a basis for our predictive analysis.

Two models were selected: LightGBM and Elastic Net. LightGBM was chosen for its training speed and efficiency, lower memory usage, and accuracy [2]. However, it can be prone to overfitting. Elastic Net was chosen due its ability to handle high dimensionality and better managing overfitting [3]. However, it isn't always suitable for data that isn't linear.

Since the COVID-19 dataset is very large and complex, these two models represent efficient ways of developing distinct models with separate advantages. The dataset will also have features added and multiple different sets to capture varying combinations of these features.

Methods

First, we need to prepare the data using the code from Kaggle. After the initial dataset is created, we will separate it into covid_train and covid_test sets. These will be used for training the model and then testing the model, respectively.

```
############################################################
# Data preparation
############################################################

# Install lightgbm and glmnet packages
if(!require(lightgbm)) install.packages("lightgbm", repos = "http://cran.us.r-project.org")
if(!require(glmnet)) install.packages("glmnet", repos = "http://cran.us.r-project.org")

# Load all required libraries
library(lightgbm)
library(glmnet)
library(httr)
library(ggplot2)
library(dplyr)

# Define the URL
url <- "https://github.com/KStroop/ChooseYourOwnProject/raw/main/corona-virus-report.zip"

# Use httr to handle the download
GET(url, write_disk("corona-virus-report.zip", overwrite = TRUE))
```

```
## Response [https://raw.githubusercontent.com/KStroop/ChooseYourOwnProject/main/corona-virus-report.
##   Date: 2024-04-27 01:05
##   Status: 200
##   Content-Type: application/zip
##   Size: 19.9 MB
## <ON DISK>  /Users/kevin.stroop/Downloads/corona-virus-report.zip
```

```
# Proceed to check the file and unzip as before
if (!file.exists("corona-virus-report.zip") || file.size("corona-virus-report.zip") == 0) {
  stop("Download failed: file is missing or empty")
}

unzip("corona-virus-report.zip", exdir = "data")

# Check if the unzipping was successful and the file exists
if (!file.exists("data/usa_county_wise.csv")) {
  stop("Unzipping failed: data/usa_county_wise.csv does not exist")
}

# Read the csv into covid_data
covid_data <- read.csv("data/usa_county_wise.csv")
```

```r
############################################################
# Data preparation
############################################################

# Ensure the Date column is in Date format
covid_data$Date <- as.Date(covid_data$Date, format = "%m/%d/%y")

# Split the data into training and testing sets
covid_train <- covid_data[covid_data$Date <= as.Date("2020-06-30"), ]
covid_test <- covid_data[covid_data$Date > as.Date("2020-06-30"), ]

# Remove the original dataset to free up memory
rm(covid_data)

# Force garbage collection to reclaim memory
gc()
```

```
##            used  (Mb) gc trigger  (Mb) limit (Mb) max used  (Mb)
## Ncells  2030850 108.5    3179630 169.9         NA  3179630 169.9
## Vcells 11103144  84.8   24534357 187.2     102400 24502401 187.0
```

```r
# Display the structure of the split datasets
str(covid_train)
```

```
## 'data.frame':    537740 obs. of  14 variables:
##  $ UID           : int  16 316 580 63072001 63072003 63072005 63072007 63072009 63072011 63072013
##  $ iso2          : chr  "AS" "GU" "MP" "PR" ...
##  $ iso3          : chr  "ASM" "GUM" "MNP" "PRI" ...
##  $ code3         : int  16 316 580 630 630 630 630 630 630 630 ...
##  $ FIPS          : num  60 66 69 72001 72003 ...
##  $ Admin2        : chr  "" "" "" "Adjuntas" ...
##  $ Province_State: chr  "American Samoa" "Guam" "Northern Mariana Islands" "Puerto Rico" ...
##  $ Country_Region: chr  "US" "US" "US" "US" ...
##  $ Lat           : num  -14.3 13.4 15.1 18.2 18.4 ...
##  $ Long_         : num  -170.1 144.8 145.7 -66.8 -67.2 ...
##  $ Combined_Key  : chr  "American Samoa, US" "Guam, US" "Northern Mariana Islands, US" "Adjuntas,
##  $ Date          : Date, format: "2020-01-22" "2020-01-22" ...
##  $ Confirmed     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Deaths        : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
str(covid_test)
```

```
## 'data.frame':    90180 obs. of  14 variables:
##  $ UID           : int  16 316 580 63072001 63072003 63072005 63072007 63072009 63072011 63072013
##  $ iso2          : chr  "AS" "GU" "MP" "PR" ...
##  $ iso3          : chr  "ASM" "GUM" "MNP" "PRI" ...
##  $ code3         : int  16 316 580 630 630 630 630 630 630 630 ...
##  $ FIPS          : num  60 66 69 72001 72003 ...
##  $ Admin2        : chr  "" "" "" "Adjuntas" ...
##  $ Province_State: chr  "American Samoa" "Guam" "Northern Mariana Islands" "Puerto Rico" ...
##  $ Country_Region: chr  "US" "US" "US" "US" ...
##  $ Lat           : num  -14.3 13.4 15.1 18.2 18.4 ...
##  $ Long_         : num  -170.1 144.8 145.7 -66.8 -67.2 ...
##  $ Combined_Key  : chr  "American Samoa, US" "Guam, US" "Northern Mariana Islands, US" "Adjuntas,
##  $ Date          : Date, format: "2020-07-01" "2020-07-01" ...
##  $ Confirmed     : int  0 267 30 50 35 55 59 42 46 128 ...
##  $ Deaths        : int  0 5 2 0 0 0 0 0 0 0 ...
```

Next we will create lag features in the data by county, which is under the Admin2 column. To ensure no errors we will replace any empty Admin2 values with "Empty".

```
############################################################
# Creating lag features and handling empty Admin2 values
############################################################

# Function to create lag features and handle empty Admin2 values
prepare_data <- function(data) {
  data %>%
    mutate(Admin2 = replace(Admin2, Admin2 == "", "Empty")) %>% # Replace empty Admin2 with
    ↪  "Empty"
    arrange(Admin2, Date) %>% # Sort by Admin2 and Date
    group_by(Admin2) %>%
    mutate(
      Confirmed_lag2 = ifelse(is.na(lag(Confirmed, 2)), 0, lag(Confirmed, 2)),
      Confirmed_lag7 = ifelse(is.na(lag(Confirmed, 7)), 0, lag(Confirmed, 7)),
      Confirmed_lag14 = ifelse(is.na(lag(Confirmed, 14)), 0, lag(Confirmed, 14))
    ) %>%
    ungroup()
}

# Apply the function to the training and testing datasets
covid_train <- prepare_data(covid_train)
covid_test <- prepare_data(covid_test)
```

```r
# Display the structure to verify the changes
str(covid_train)
```

```
## tibble [537,740 x 17] (S3: tbl_df/tbl/data.frame)
##  $ UID            : int [1:537740] 84045001 84045001 84045001 84045001 84045001 84045001 84045001
##  $ iso2           : chr [1:537740] "US" "US" "US" "US" ...
##  $ iso3           : chr [1:537740] "USA" "USA" "USA" "USA" ...
##  $ code3          : int [1:537740] 840 840 840 840 840 840 840 840 840 840 ...
##  $ FIPS           : num [1:537740] 45001 45001 45001 45001 45001 ...
##  $ Admin2         : chr [1:537740] "Abbeville" "Abbeville" "Abbeville" "Abbeville" ...
##  $ Province_State : chr [1:537740] "South Carolina" "South Carolina" "South Carolina" "South Carol
##  $ Country_Region : chr [1:537740] "US" "US" "US" "US" ...
##  $ Lat            : num [1:537740] 34.2 34.2 34.2 34.2 34.2 ...
##  $ Long_          : num [1:537740] -82.5 -82.5 -82.5 -82.5 -82.5 ...
##  $ Combined_Key   : chr [1:537740] "Abbeville, South Carolina, US" "Abbeville, South Carolina, US'
##  $ Date           : Date[1:537740], format: "2020-01-22" "2020-01-23" ...
##  $ Confirmed      : int [1:537740] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Deaths         : int [1:537740] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Confirmed_lag2 : num [1:537740] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Confirmed_lag7 : num [1:537740] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Confirmed_lag14: num [1:537740] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
str(covid_test)
```

```
## tibble [90,180 x 17] (S3: tbl_df/tbl/data.frame)
##  $ UID            : int [1:90180] 84045001 84045001 84045001 84045001 84045001 84045001 84045001 8
##  $ iso2           : chr [1:90180] "US" "US" "US" "US" ...
##  $ iso3           : chr [1:90180] "USA" "USA" "USA" "USA" ...
##  $ code3          : int [1:90180] 840 840 840 840 840 840 840 840 840 840 ...
##  $ FIPS           : num [1:90180] 45001 45001 45001 45001 45001 ...
##  $ Admin2         : chr [1:90180] "Abbeville" "Abbeville" "Abbeville" "Abbeville" ...
##  $ Province_State : chr [1:90180] "South Carolina" "South Carolina" "South Carolina" "South Caroli
##  $ Country_Region : chr [1:90180] "US" "US" "US" "US" ...
##  $ Lat            : num [1:90180] 34.2 34.2 34.2 34.2 34.2 ...
##  $ Long_          : num [1:90180] -82.5 -82.5 -82.5 -82.5 -82.5 ...
##  $ Combined_Key   : chr [1:90180] "Abbeville, South Carolina, US" "Abbeville, South Carolina, US"
##  $ Date           : Date[1:90180], format: "2020-07-01" "2020-07-02" ...
##  $ Confirmed      : int [1:90180] 113 119 118 119 124 135 134 137 138 141 ...
##  $ Deaths         : int [1:90180] 0 0 0 0 0 0 1 1 1 1 ...
##  $ Confirmed_lag2 : num [1:90180] 0 0 113 119 118 119 124 135 134 137 ...
##  $ Confirmed_lag7 : num [1:90180] 0 0 0 0 0 0 0 113 119 118 ...
##  $ Confirmed_lag14: num [1:90180] 0 0 0 0 0 0 0 0 0 0 ...
```

Then we will do the same process to create rolling averages.

```
############################################################
# Calculating rolling averages
############################################################

# Calculate rolling averages for the training set
covid_train <- covid_train %>%
  group_by(Admin2) %>%
  arrange(Admin2, Date, .by_group = TRUE) %>%
  mutate(
    RollingAvg7 = zoo::rollapply(Confirmed, 7, mean, fill = 0, align = "right"),
    RollingAvg14 = zoo::rollapply(Confirmed, 14, mean, fill = 0, align = "right"),
    RollingAvg30 = zoo::rollapply(Confirmed, 30, mean, fill = 0, align = "right")
  ) %>%
  ungroup()

# Calculate rolling averages for the testing set
covid_test <- covid_test %>%
  group_by(Admin2) %>%
  arrange(Admin2, Date, .by_group = TRUE) %>%
  mutate(
    RollingAvg7 = zoo::rollapply(Confirmed, 7, mean, fill = 0, align = "right"),
    RollingAvg14 = zoo::rollapply(Confirmed, 14, mean, fill = 0, align = "right"),
    RollingAvg30 = zoo::rollapply(Confirmed, 30, mean, fill = 0, align = "right")
  ) %>%
  ungroup()
```

Then we will do the same process to create growth rates.

```
############################################################
# Calculating growth rates
############################################################

# Making sure that there are no Inf or -Inf values generated
calculate_growth_rate <- function(current, previous) {
  if (is.na(current) || is.na(previous) || previous == 0) {
    return(0)
  } else {
    return((current - previous) / previous)
  }
}

# Calculate growth rates for the training set
covid_train <- covid_train %>%
  group_by(Admin2) %>%
  arrange(Admin2, Date, .by_group = TRUE) %>%
  mutate(
    GrowthRate1 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 1)),
    GrowthRate7 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 7)),
    GrowthRate14 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 14)),
    GrowthRate30 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 30))
  ) %>%
  ungroup()

# Calculate growth rates for the testing set
covid_test <- covid_test %>%
  group_by(Admin2) %>%
  arrange(Admin2, Date, .by_group = TRUE) %>%
  mutate(
    GrowthRate1 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 1)),
    GrowthRate7 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 7)),
    GrowthRate14 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 14)),
    GrowthRate30 = mapply(calculate_growth_rate, Confirmed, lag(Confirmed, 30))
  ) %>%
  ungroup()
```

We then need to check for missing values since it is possible that some of the created features could introduce missing values.

```
############################################################
# Checking for NA, NaN, Inf, -Inf values
############################################################

# Function to summarize NA, NaN, Inf, and -Inf values in each column
summarize_na_nan_inf <- function(data) {
  summary_data <- data.frame(
    Column = character(),
    NA_Count = integer(),
    NaN_Count = integer(),
    Inf_Count = integer(),
    Neg_Inf_Count = integer(),
    stringsAsFactors = FALSE
  )

  for (column in colnames(data)) {
    na_count <- sum(is.na(data[[column]]))
    nan_count <- sum(is.nan(data[[column]]))
    inf_count <- sum(data[[column]] == Inf)
    neg_inf_count <- sum(data[[column]] == -Inf)

    summary_data <- rbind(summary_data, data.frame(
      Column = column,
      NA_Count = na_count,
      NaN_Count = nan_count,
      Inf_Count = inf_count,
      Neg_Inf_Count = neg_inf_count
    ))
  }

  return(summary_data)
}

# Apply the function to covid_train and covid_test
train_summary <- summarize_na_nan_inf(covid_train)
test_summary <- summarize_na_nan_inf(covid_test)
```

```r
############################################################
# Print the train summary
############################################################

print(train_summary)
```

```
##             Column NA_Count NaN_Count Inf_Count Neg_Inf_Count
## 1              UID        0         0         0             0
## 2             iso2        0         0         0             0
## 3             iso3        0         0         0             0
## 4            code3        0         0         0             0
## 5             FIPS     1610         0        NA            NA
## 6           Admin2        0         0         0             0
## 7   Province_State        0         0         0             0
## 8   Country_Region        0         0         0             0
## 9              Lat        0         0         0             0
## 10           Long_        0         0         0             0
## 11    Combined_Key        0         0         0             0
## 12            Date        0         0         0             0
## 13       Confirmed        0         0         0             0
## 14          Deaths        0         0         0             0
## 15  Confirmed_lag2        0         0         0             0
## 16  Confirmed_lag7        0         0         0             0
## 17 Confirmed_lag14        0         0         0             0
## 18     RollingAvg7        0         0         0             0
## 19    RollingAvg14        0         0         0             0
## 20    RollingAvg30        0         0         0             0
## 21     GrowthRate1        0         0         0             0
## 22     GrowthRate7        0         0         0             0
## 23    GrowthRate14        0         0         0             0
## 24    GrowthRate30        0         0         0             0
```

```
############################################################
# Print the test summary
############################################################

print(test_summary)
```

```
##             Column NA_Count NaN_Count Inf_Count Neg_Inf_Count
## 1              UID        0         0         0             0
## 2             iso2        0         0         0             0
## 3             iso3        0         0         0             0
## 4            code3        0         0         0             0
## 5             FIPS      270         0        NA            NA
## 6           Admin2        0         0         0             0
## 7   Province_State        0         0         0             0
## 8   Country_Region        0         0         0             0
## 9              Lat        0         0         0             0
## 10           Long_        0         0         0             0
## 11    Combined_Key        0         0         0             0
## 12            Date        0         0         0             0
## 13       Confirmed        0         0         0             0
## 14          Deaths        0         0         0             0
## 15  Confirmed_lag2        0         0         0             0
## 16  Confirmed_lag7        0         0         0             0
## 17 Confirmed_lag14        0         0         0             0
## 18     RollingAvg7        0         0         0             0
## 19    RollingAvg14        0         0         0             0
## 20    RollingAvg30        0         0         0             0
## 21     GrowthRate1        0         0         0             0
## 22     GrowthRate7        0         0         0             0
## 23    GrowthRate14        0         0         0             0
## 24    GrowthRate30        0         0         0             0
```

Since FIPS was the only with empty values we will set those empty values to 0. FIPS will not be used in our analysis, but it is best to ensure there are no empty values anyway.

```
############################################################
# Handling NA and NaN values
############################################################

covid_train$FIPS[is.na(covid_train$FIPS)] <- 0
covid_test$FIPS[is.na(covid_test$FIPS)] <- 0
```

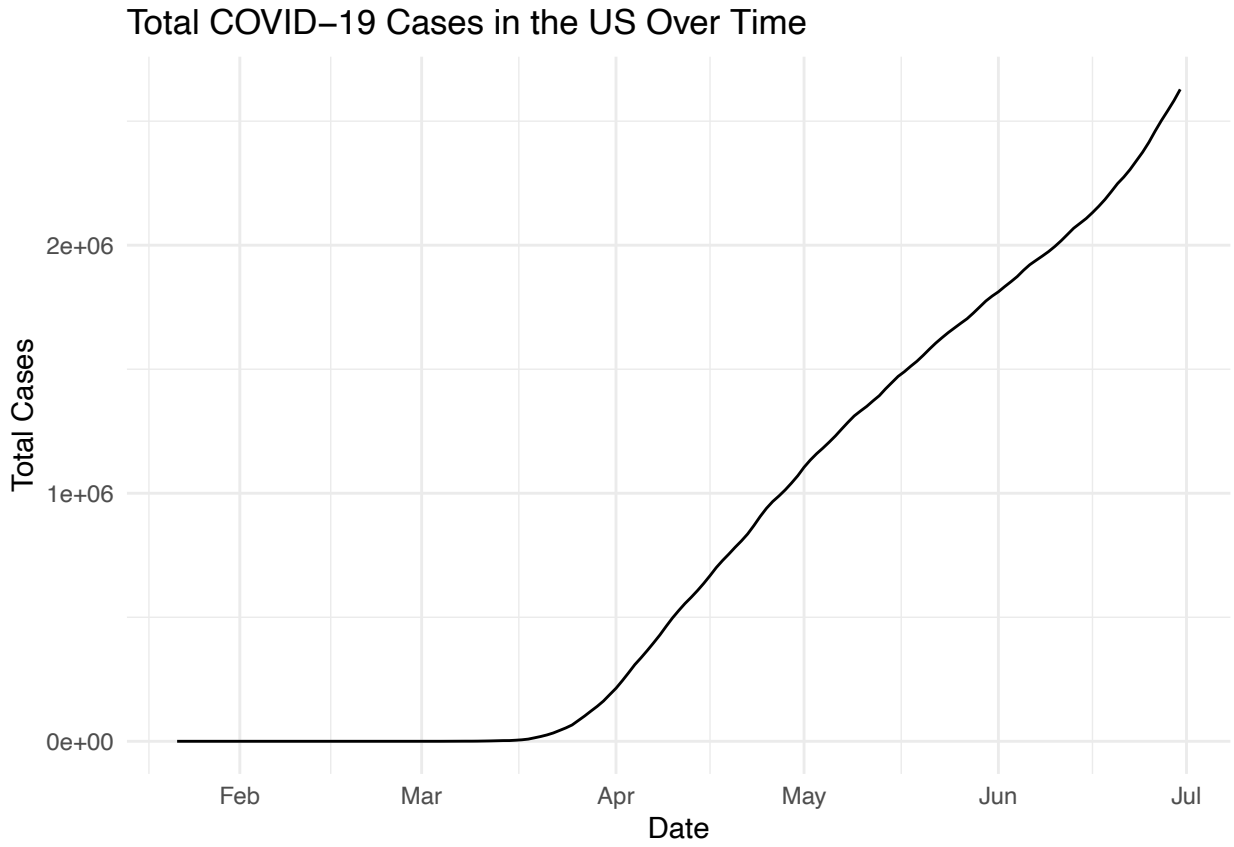
For our model we will only be considering continental US so we will filter the training and testing sets by states within the continental U.S.

```
############################################################
# Filtering for continental US
############################################################

continental_us_states <- c("Alabama", "Alaska", "Arizona", "Arkansas", "California",
↪  "Colorado",
                           "Connecticut", "Delaware", "Florida", "Georgia", "Hawaii", "Idaho",
                           ↪
                           "Illinois", "Indiana", "Iowa", "Kansas", "Kentucky", "Louisiana",
                           "Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota",
                           "Mississippi", "Missouri", "Montana", "Nebraska", "Nevada",
                           "New Hampshire", "New Jersey", "New Mexico", "New York",
                           "North Carolina", "North Dakota", "Ohio", "Oklahoma", "Oregon",
                           "Pennsylvania", "Rhode Island", "South Carolina", "South Dakota",
                           "Tennessee", "Texas", "Utah", "Vermont", "Virginia", "Washington",
                           "West Virginia", "Wisconsin", "Wyoming", "District of Columbia")

covid_train <- covid_train %>% filter(Province_State %in% continental_us_states)
covid_test <- covid_test %>% filter(Province_State %in% continental_us_states)
```

Data Graphing

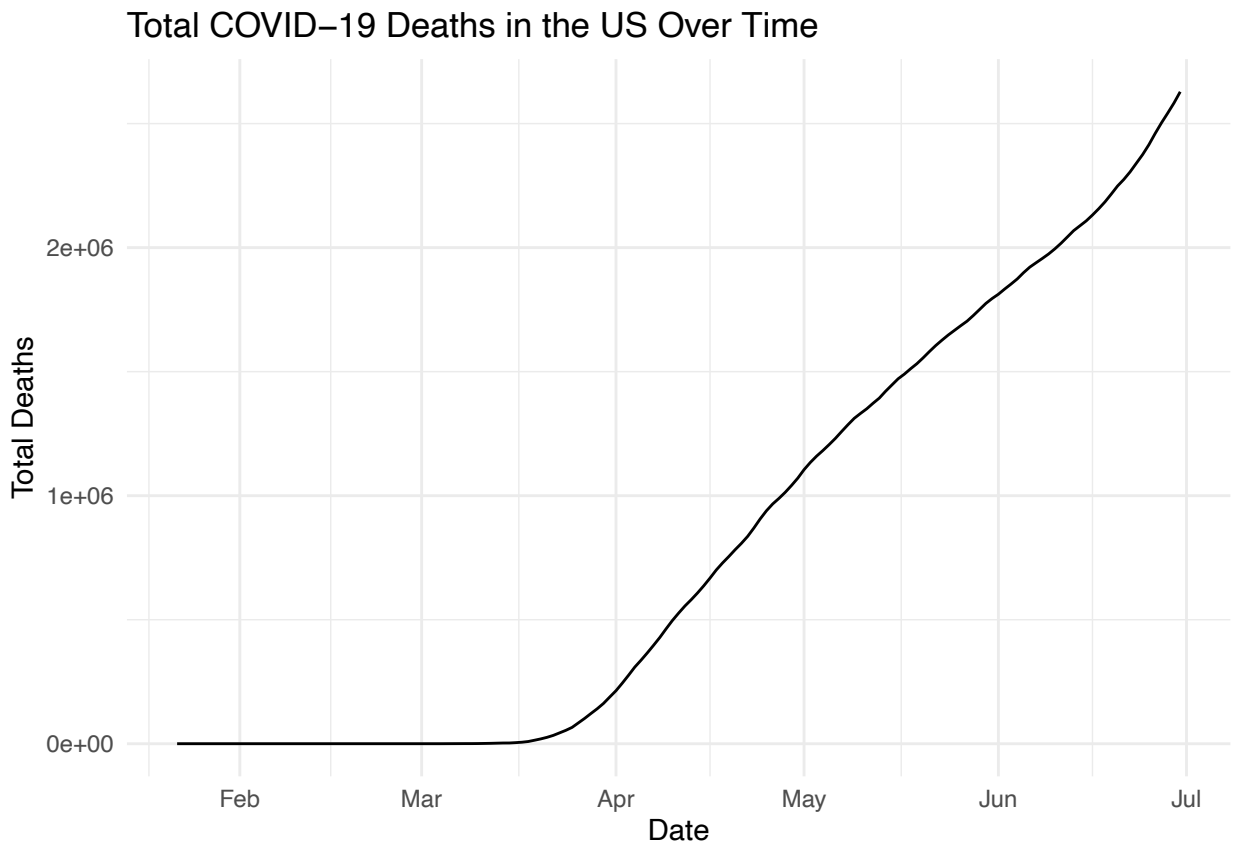
First we will graph the cases per day.

```
############################################################
# Graphing cases
############################################################

# Convert the Date column to a Date object
covid_train$Date <- as.Date(covid_train$Date, format = "%m/%d/%y")

# Summing the total cases per day
total_cases_per_day <- covid_train %>%
  group_by(Date) %>%
  summarise(Total_Cases = sum(Confirmed, na.rm = TRUE))  # Total cases per day

# Creating the plot for daily total cases
plot <- ggplot(total_cases_per_day, aes(x = Date, y = Total_Cases)) +
  geom_line() +  # Plot the total cases as a line
  labs(title = "Total COVID-19 Cases in the US Over Time",
       x = "Date",
       y = "Total Cases") +
  theme_minimal()

# Displaying the plot
print(plot)
```

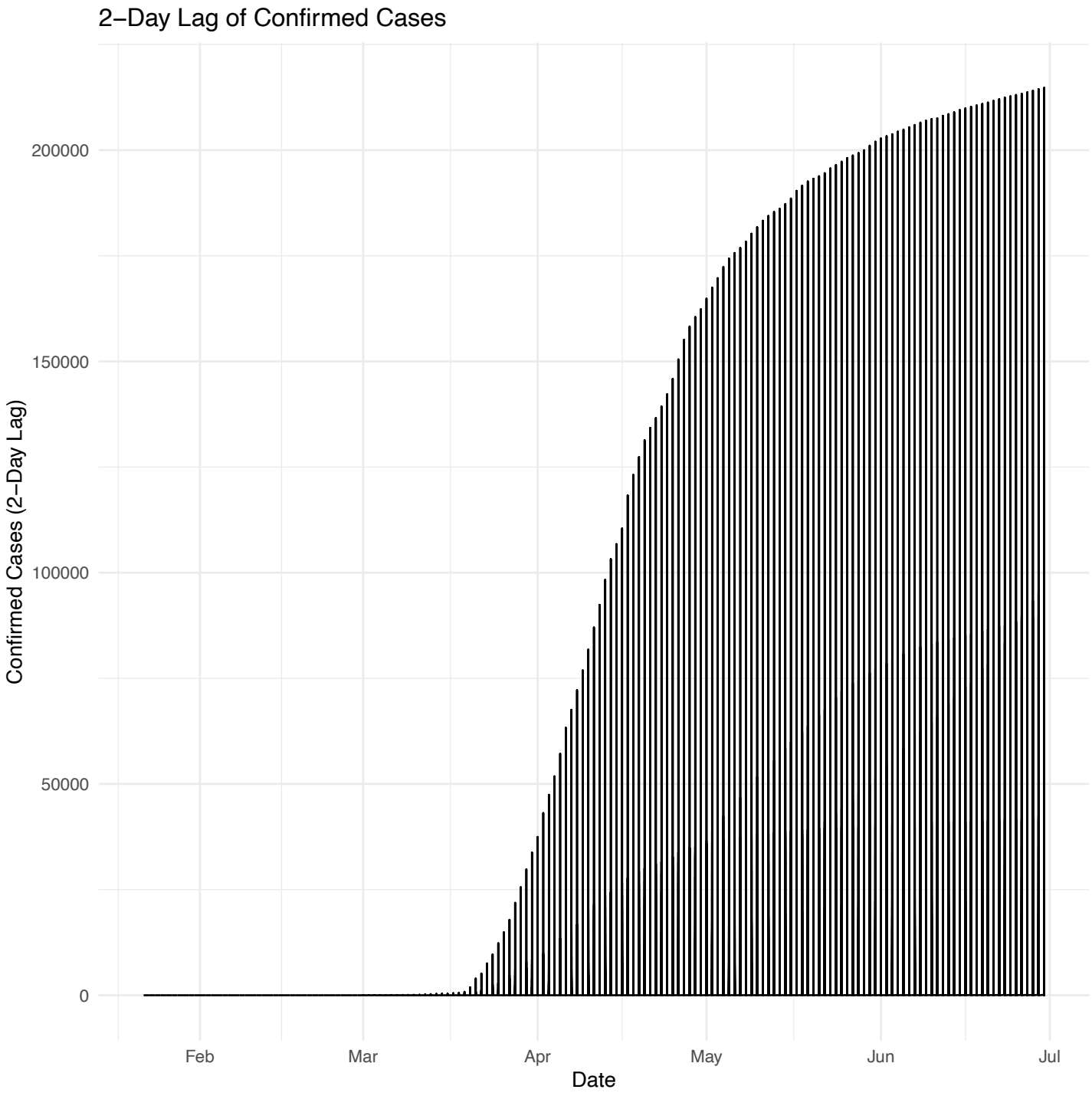


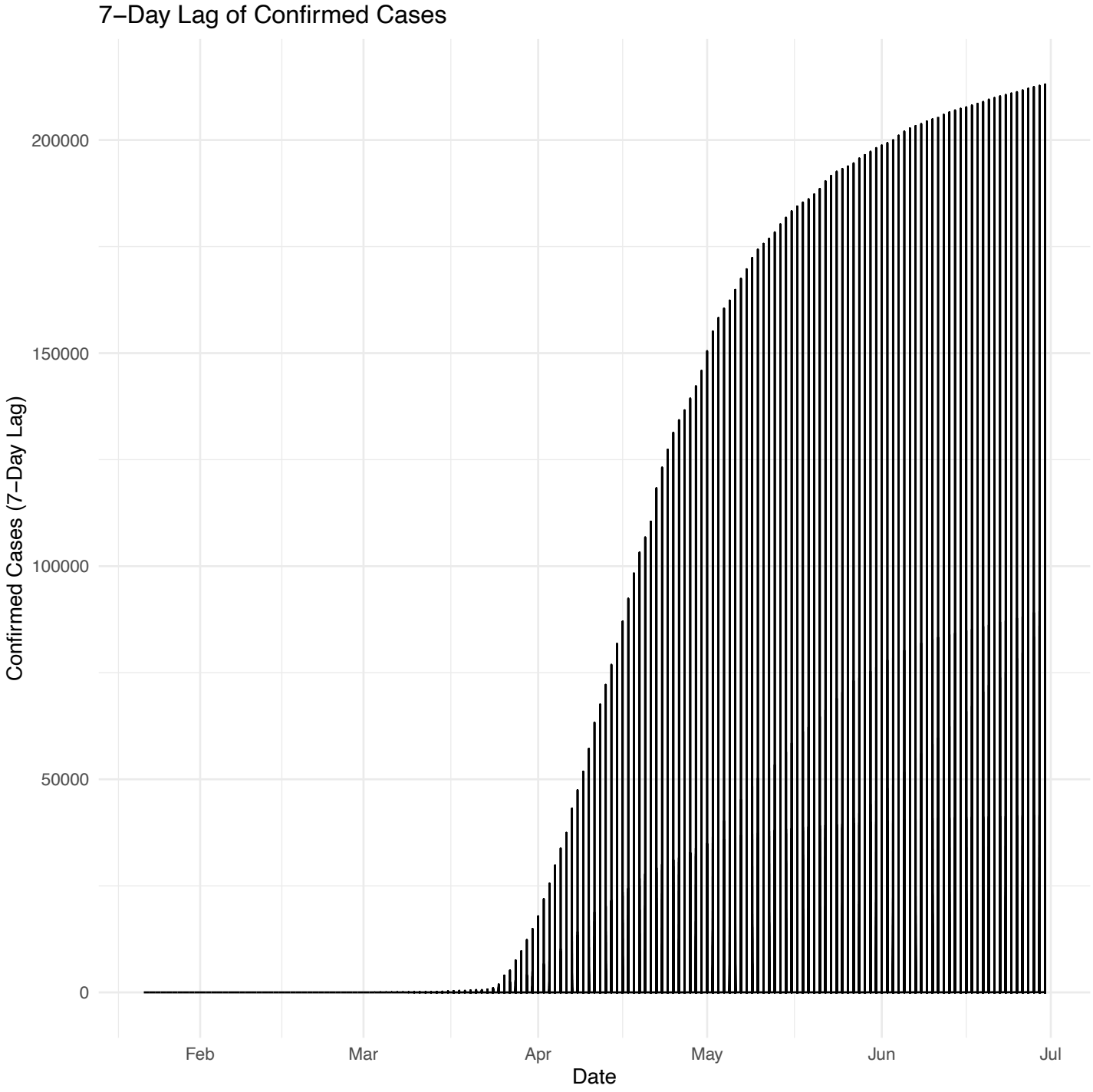
Second we will graph the deaths per day.

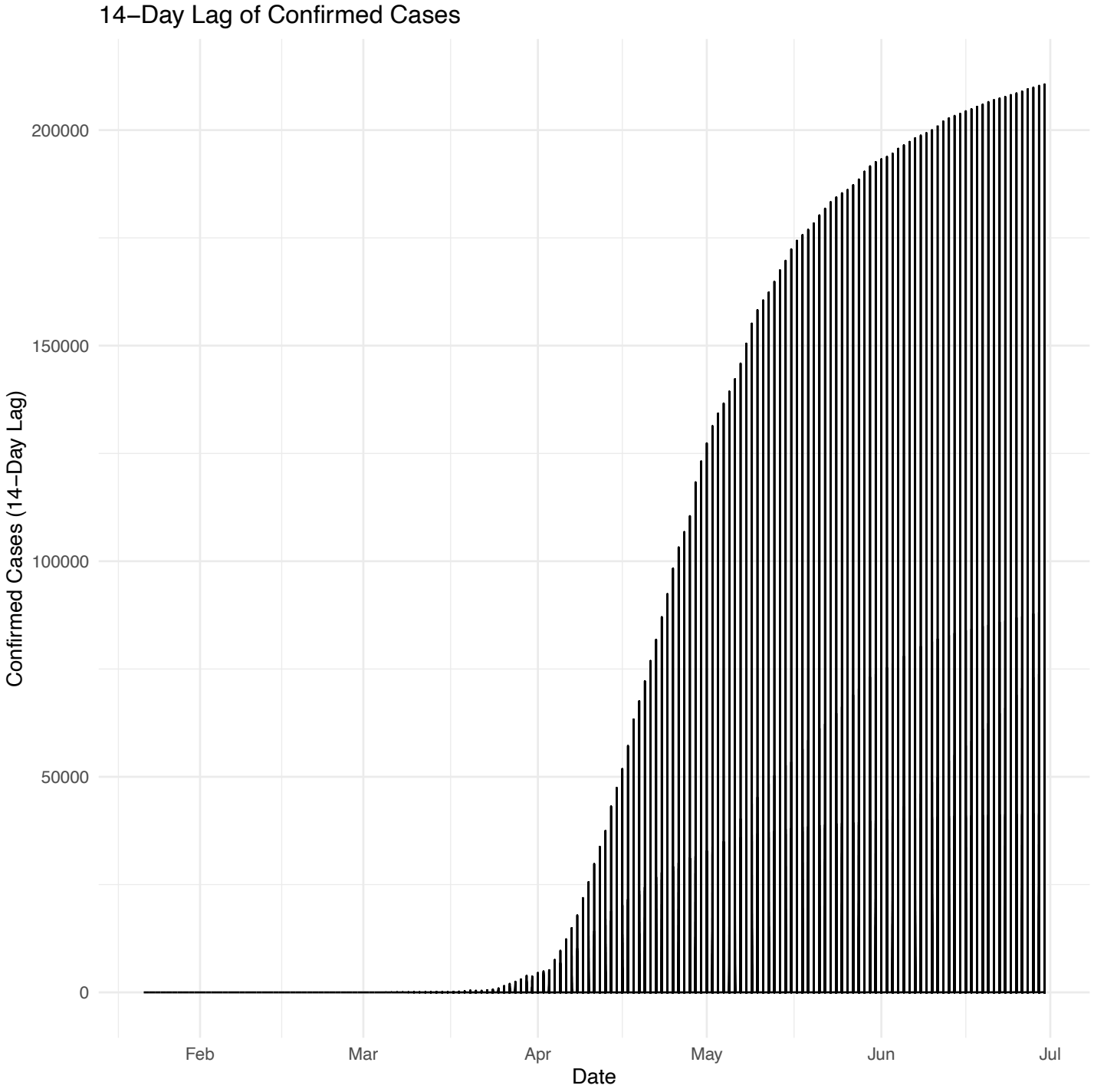
```
############################################################
# Graphing deaths
############################################################

# Convert the Date column to a Date object
covid_train$Date <- as.Date(covid_train$Date, format = "%m/%d/%y")

# Summing the total deaths per day
total_deaths_per_day <- covid_train %>%
  group_by(Date) %>%
  summarise(Total_Deaths = sum(Confirmed, na.rm = TRUE))  # Total deaths per day

# Creating the plot for daily total deaths
plot <- ggplot(total_deaths_per_day, aes(x = Date, y = Total_Deaths)) +
  geom_line() +  # Plot the total deaths as a line
  labs(title = "Total COVID-19 Deaths in the US Over Time",
       x = "Date",
       y = "Total Deaths") +
  theme_minimal()

# Displaying the plot
print(plot)
```



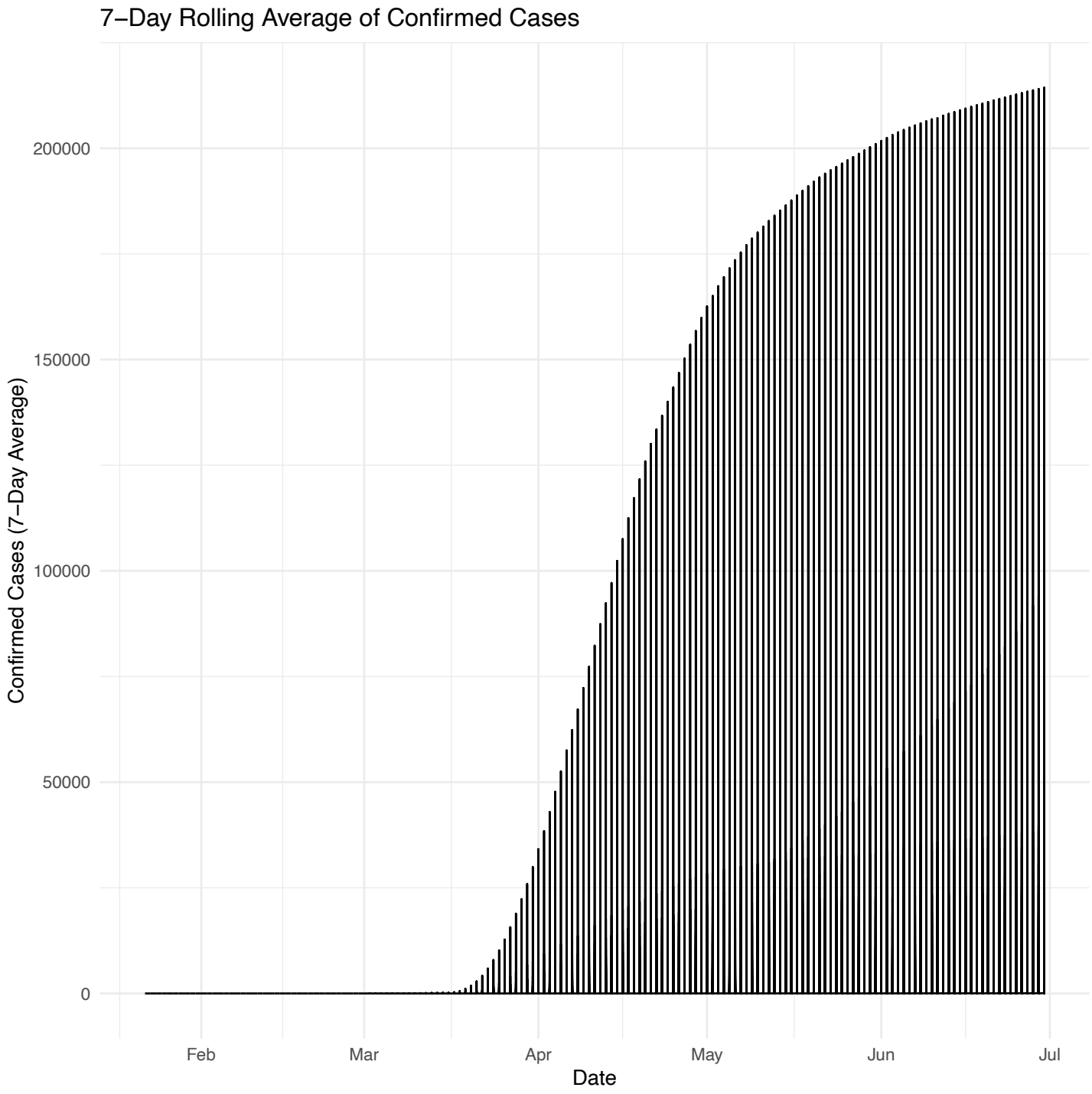
Third we will graph the lag in cases.

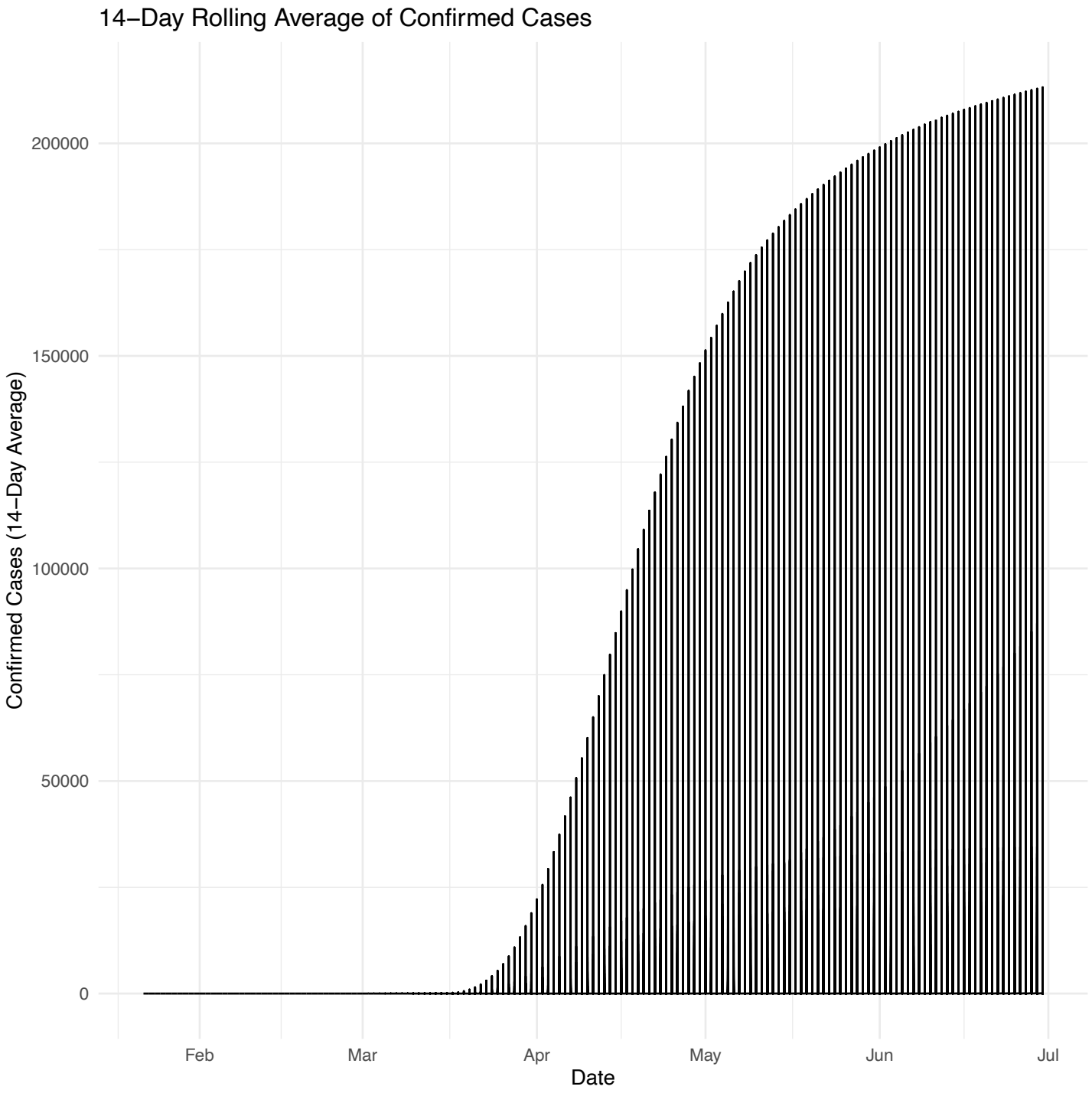


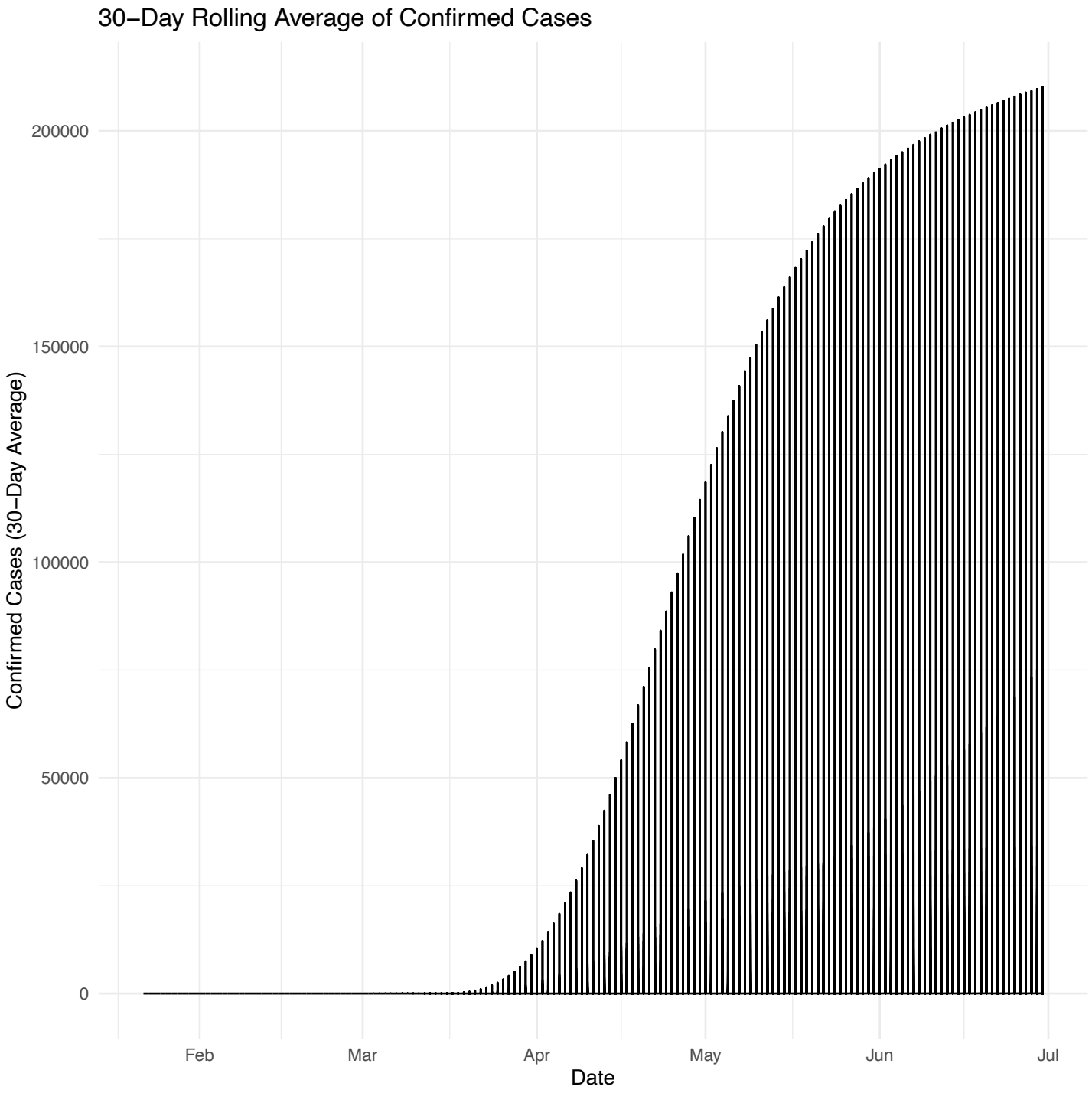

7−Day Lag of Confirmed Cases
200000
150000
100000
50000
0
Confirmed Cases (7−Day Lag)
Feb
Mar
Apr
May
Jun
Jul
Date


14–Day Lag of Confirmed Cases
Confirmed Cases (14–Day Lag)
200000
150000
100000
50000
0
Feb
Mar
Apr
May
Jun
Jul
Date

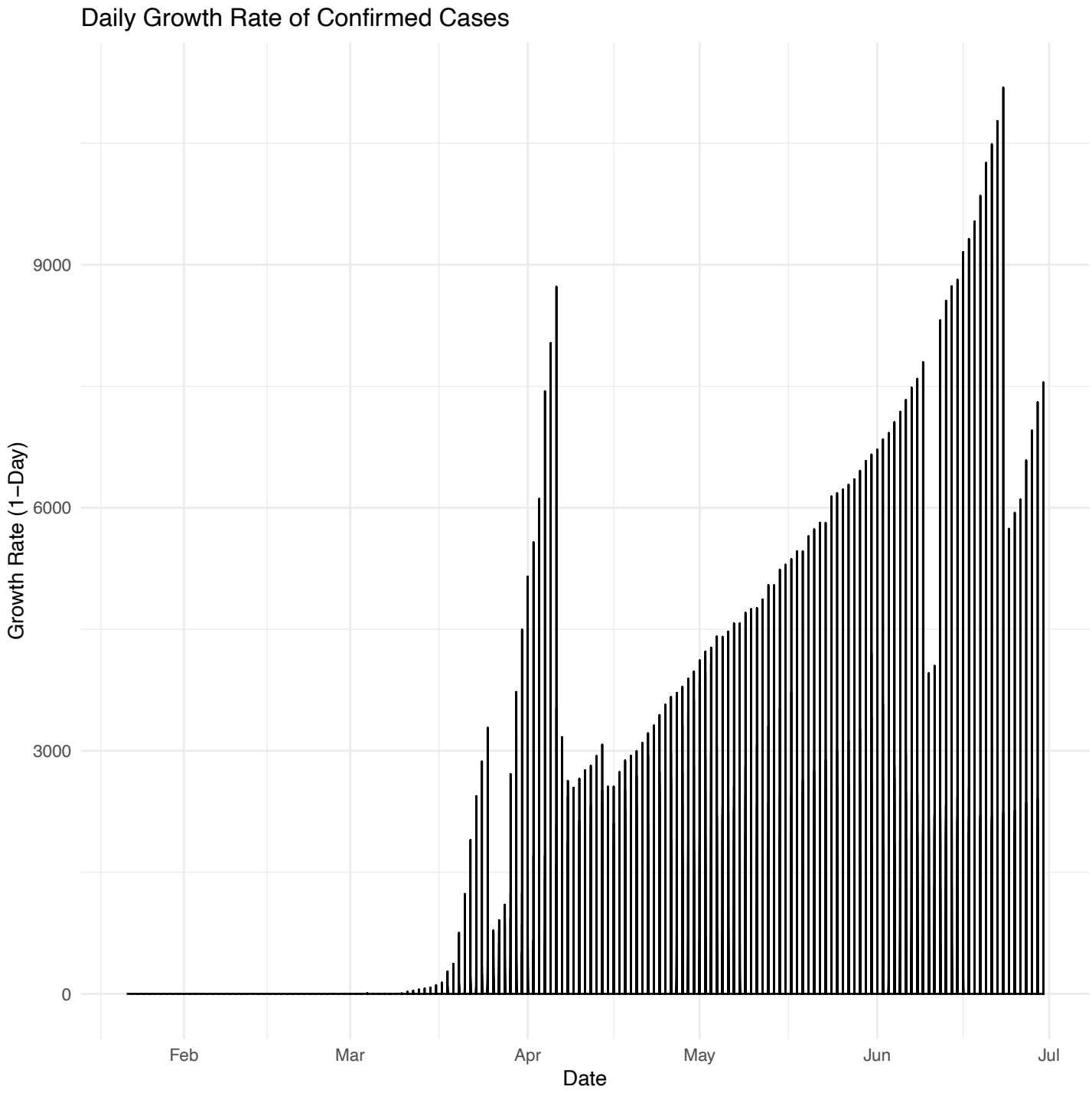
Fourth we will graph the rolling averages.

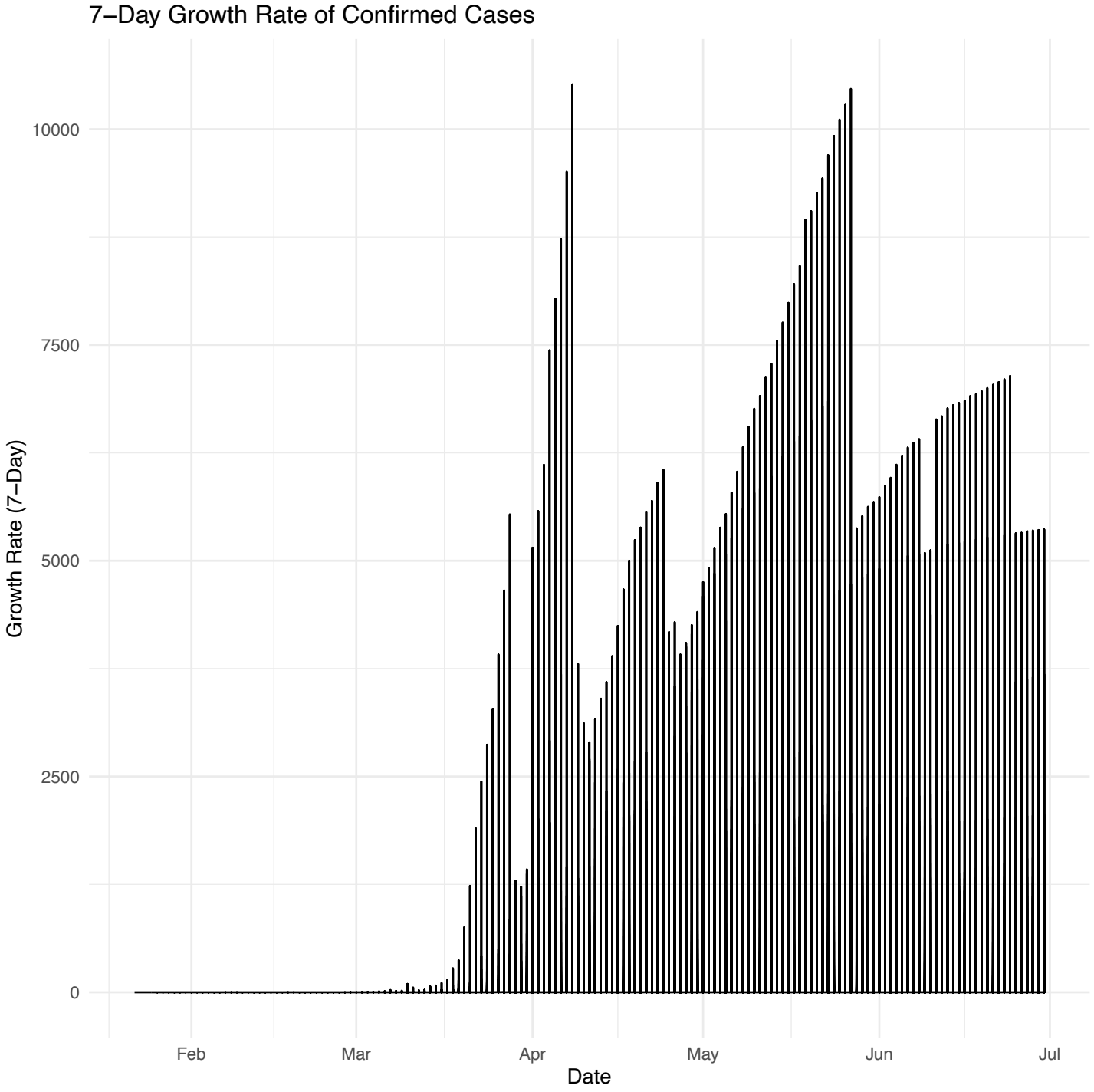


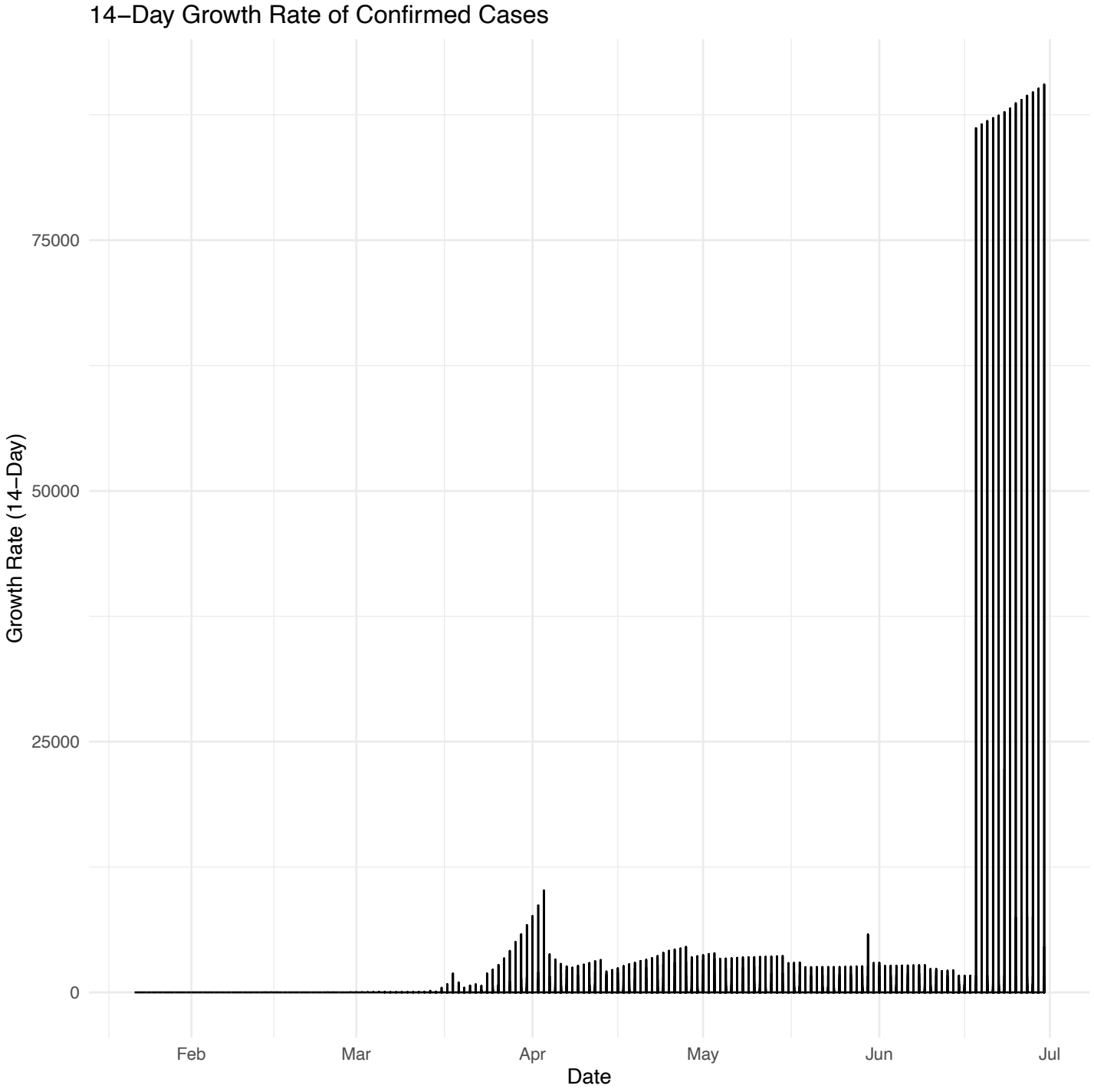

14−Day Rolling Average of Confirmed Cases
Confirmed Cases (14−Day Average)
200000
150000
100000
50000
0
Feb
Mar
Apr
May
Jun
Jul
Date

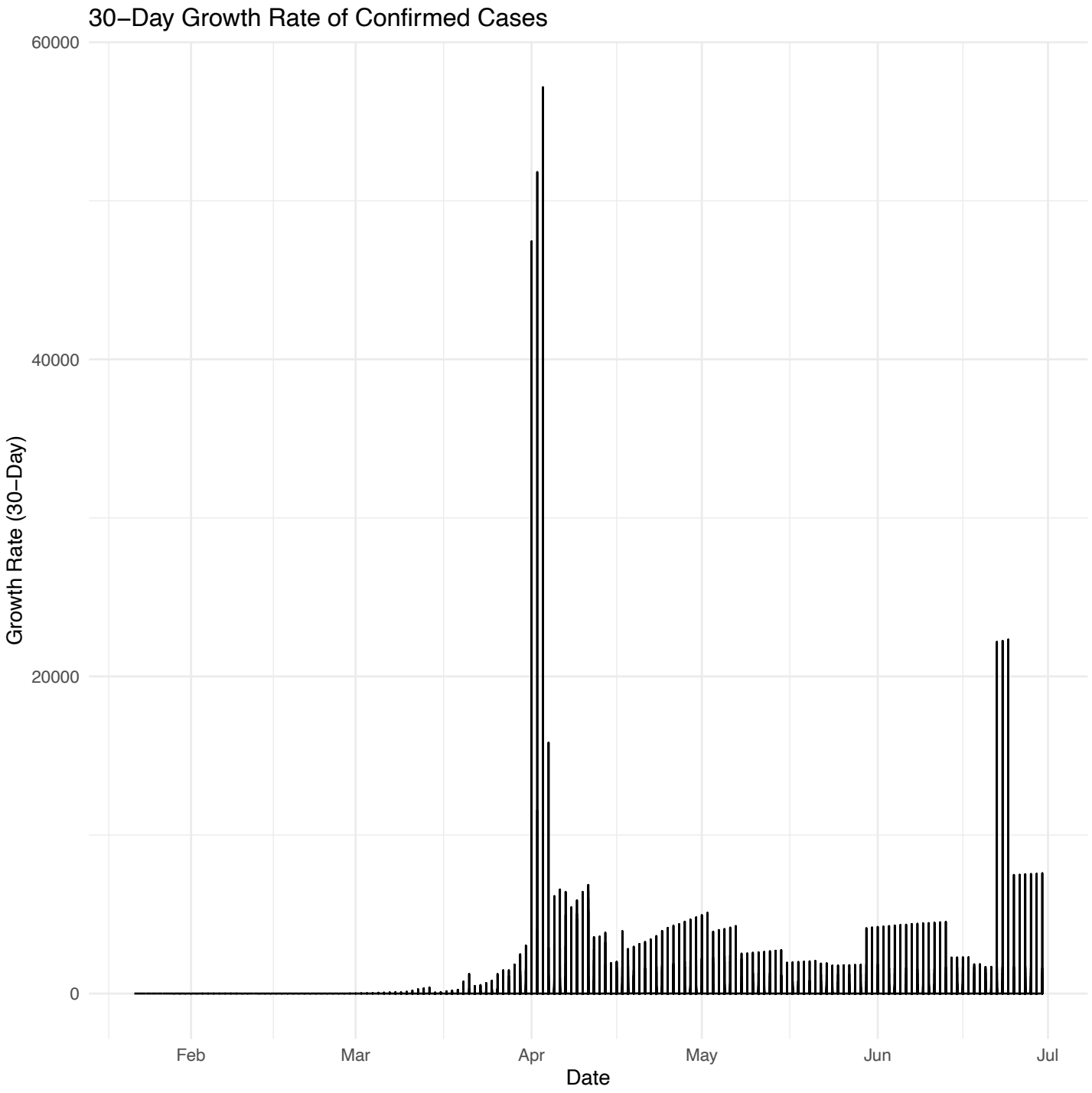

30−Day Rolling Average of Confirmed Cases
Confirmed Cases (30−Day Average)
200000
150000
100000
50000
0
Feb
Mar
Apr
May
Jun
Jul
Date

Finally we will graph the growth rates.




7−Day Growth Rate of Confirmed Cases
Growth Rate (7−Day)
10000
7500
5000
2500
0
Feb
Mar
Apr
May
Jun
Jul
Date


14–Day Growth Rate of Confirmed Cases
Growth Rate (14–Day)
75000
50000
25000
0
Feb
Mar
Apr
May
Jun
Jul
Date


30−Day Growth Rate of Confirmed Cases
60000
40000
20000
0
Growth Rate (30−Day)
Feb
Mar
Apr
May
Jun
Jul
Date

Preparing feature and target sets for training

Before training the model we need to convert the states into numerical values using one-hot encoding.

```
############################################################
# One-hot encoding
############################################################

# Ensure Province_State is a factor with consistent levels in both datasets
unique_states <- unique(c(covid_train$Province_State, covid_test$Province_State))
covid_train$Province_State <- factor(covid_train$Province_State, levels = unique_states)
covid_test$Province_State <- factor(covid_test$Province_State, levels = unique_states)

# Creating dummy variables for the training set
OHE_PS_train <- model.matrix(~ Province_State - 1, data = covid_train)
covid_train <- cbind(covid_train, OHE_PS_train)

# Creating dummy variables for the testing set
OHE_PS_test <- model.matrix(~ Province_State - 1, data = covid_test)
covid_test <- cbind(covid_test, OHE_PS_test)
```

We will prepare the feature and target sets for training and testing, which will include the lag features, rolling averages, and growth rates previously generated. We will exclude Confirmed from X because it is the target variable which will be included in y.

Because we have created 3 features and each of these have their own parts, we will make sets including and excluding every combination for a total of 8 sets as noted below:

Set 1: Base dataset with no added features
Set 2: Includes the lag feature
Set 3: Includes the rolling average feature
Set 4: Includes the growth rate feature
Set 5: Includes the lag and rolling average features
Set 6: Includes the lag and growth rate features
Set 7: Includes the rolling average and growth rate features
Set 8: Includes all three added features

Code will use the number in the list above to distinguish between sets.

```r
############################################################
# Prepare feature sets
############################################################
# Set 1: Base dataset with no added features
X_train_1 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -RollingAvg7, -RollingAvg14, -RollingAvg30,
↪  -GrowthRate1, -GrowthRate7, -GrowthRate14, -GrowthRate30))
X_test_1 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -RollingAvg7, -RollingAvg14, -RollingAvg30,
↪  -GrowthRate1, -GrowthRate7, -GrowthRate14, -GrowthRate30))

# Set 2: Including the lag feature
X_train_2 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -RollingAvg7,
↪  -RollingAvg14, -RollingAvg30, -GrowthRate1, -GrowthRate7, -GrowthRate14, -GrowthRate30))
X_test_2 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -RollingAvg7,
↪  -RollingAvg14, -RollingAvg30, -GrowthRate1, -GrowthRate7, -GrowthRate14, -GrowthRate30))

# Set 3: Including the rolling average feature
X_train_3 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -GrowthRate1, -GrowthRate7, -GrowthRate14,
↪  -GrowthRate30))
X_test_3 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -GrowthRate1, -GrowthRate7, -GrowthRate14,
↪  -GrowthRate30))

# Set 4: Including the growth rate feature
X_train_4 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -RollingAvg7, -RollingAvg14, -RollingAvg30))
X_test_4 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14, -RollingAvg7, -RollingAvg14, -RollingAvg30))

# Set 5: Including the lag and rolling average features
X_train_5 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -GrowthRate1,
↪  -GrowthRate7, -GrowthRate14, -GrowthRate30))
X_test_5 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -GrowthRate1,
↪  -GrowthRate7, -GrowthRate14, -GrowthRate30))

# Set 6: Including the lag and growth rate features
X_train_6 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -RollingAvg7,
↪  -RollingAvg14, -RollingAvg30))
X_test_6 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -RollingAvg7,
↪  -RollingAvg14, -RollingAvg30))

# Set 7: Including the rolling average and growth rate features
X_train_7 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14))
X_test_7 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2, -Confirmed_lag2,
↪  -Confirmed_lag7, -Confirmed_lag14))

# Set 8: Including the lag, rolling average, and growth rate features
X_train_8 <- as.matrix(covid_train %>% select(-Confirmed, -FIPS, -Admin2))
X_test_8 <- as.matrix(covid_test %>% select(-Confirmed, -FIPS, -Admin2))
```

```r
############################################################
# Prepare target variables
############################################################

# Setting target variable for all sets
y_train <- covid_train$Confirmed
y_test <- covid_test$Confirmed

# Function to check if certain features are present in each set
check_features <- function(dataset, features) {
  feature_presence <- sapply(features, function(feature) {
    sum(grepl(feature, colnames(dataset))) > 0
  })
  names(feature_presence) <- features
  return(feature_presence)
}

# Define the feature patterns to look for
features_to_check <- c("Confirmed_lag", "RollingAvg", "GrowthRate")

# Initialize a data frame to store the presence of features in each set
feature_presence_results <- data.frame(
  Set = character(),
  Includes_Lag = logical(),
  Includes_Rolling_Avg = logical(),
  Includes_Growth_Rate = logical(),
  stringsAsFactors = FALSE
)

# Check the feature presence for each set
for (i in 1:8) {
  X_train_var <- get(paste0("X_train_", i))
  X_test_var <- get(paste0("X_test_", i))

  # Check features in training set
  train_features_present <- check_features(X_train_var, features_to_check)

  # Check features in test set
  test_features_present <- check_features(X_test_var, features_to_check)

  # Assuming the same features are present in both train and test sets, we can just use
  ↪  train_features_present
  feature_presence_results <- rbind(feature_presence_results, data.frame(
    Set = paste0("Set ", i),
    Includes_Lag = train_features_present["Confirmed_lag"],
    Includes_Rolling_Avg = train_features_present["RollingAvg"],
    Includes_Growth_Rate = train_features_present["GrowthRate"]
  ))
}
```

```r
############################################################
# Printing the feature table
############################################################

# Reset row names to ensure they are sequential
row.names(feature_presence_results) <- NULL

# Print the table of feature presence results
print(feature_presence_results)
```

```
##     Set Includes_Lag Includes_Rolling_Avg Includes_Growth_Rate
## 1 Set 1        FALSE                FALSE                FALSE
## 2 Set 2         TRUE                FALSE                FALSE
## 3 Set 3        FALSE                 TRUE                FALSE
## 4 Set 4        FALSE                FALSE                 TRUE
## 5 Set 5         TRUE                 TRUE                FALSE
## 6 Set 6         TRUE                FALSE                 TRUE
## 7 Set 7        FALSE                 TRUE                 TRUE
## 8 Set 8         TRUE                 TRUE                 TRUE
```

Training and Testing LightGBM and Elastic Net Models

Finally, we will train the LightGBM and Elastic Net models. These will be run to obtain RMSE scores, but also normalized RMSE scores to better determine how close to the real data in the test set the each model and each set is. The best combination of model and set will then be identified and selected.

```r
############################################################
# Running LightGBM model
############################################################
# Initialize a data frame to store the RMSE results
rmse_results <- data.frame(
  Set = paste0("Set ", 1:8),
  LightGBM = rep(NA, 8), # Placeholder vector of NAs
  ElasticNet = rep(NA, 8), # Placeholder vector of NAs
  stringsAsFactors = FALSE
)

# Iterate over each feature set
for (i in 1:8) {
  # Dynamically construct the training and testing set variable names
  X_train_var <- get(paste0("X_train_", i))
  X_test_var <- get(paste0("X_test_", i))

  # Convert data to LightGBM format
  train_data <- lgb.Dataset(data = X_train_var, label = y_train)
  test_data <- lgb.Dataset(data = X_test_var, label = y_test, reference = train_data)

  # Set parameters for the model
  params <- list(
    objective = "regression",
    metric = "rmse",
    num_leaves = 31,
    learning_rate = 0.05,
    n_estimators = 100
  )

  # Train the model
  lgb_model <- lgb.train(
    params = params,
    data = train_data,
    nrounds = 100,
    valids = list(test = test_data),
    early_stopping_rounds = 10,
    verbose = -1
  )

  # Make predictions and calculate RMSE for the test set
  predictions_lgb <- predict(lgb_model, X_test_var)
  rmse_lgbm <- sqrt(mean((predictions_lgb - y_test)^2))

  # Add the results to the data frame
  rmse_results$LightGBM[i] <- rmse_lgbm
}
```

```r
############################################################
# Running Elastic Net model
############################################################

# Iterate over each feature set
for (i in 1:8) {
  # Dynamically construct the training and testing set variable names
  X_train_var <- get(paste0("X_train_", i))
  X_test_var <- get(paste0("X_test_", i))

  ############################################################
  # Elastic Net Model
  ############################################################
  # Prepare matrix as glmnet requires a matrix
  X_train_matrix <- X_train_var
  X_test_matrix <- X_test_var

  # Alpha = 0.5 for elastic net (0 for ridge, 1 for lasso)
  alpha_value <- 0.5

  # Fit the model
  cv_model <- cv.glmnet(X_train_matrix, y_train, alpha = alpha_value, family = "gaussian")

  # Best lambda value
  best_lambda <- cv_model$lambda.min

  # Refit the model with the best lambda value
  final_model <- glmnet(X_train_matrix, y_train, alpha = alpha_value, lambda = best_lambda,
↪  family = "gaussian")

  # Predict on test set
  predictions_enet <- predict(final_model, s = best_lambda, newx = X_test_matrix)

  # Calculate RMSE for Elastic Net
  rmse_enet <- sqrt(mean((y_test - predictions_enet)^2))

  # Add the results to the data frame
  rmse_results$ElasticNet[i] <- rmse_enet
}
```

Results

```
############################################################
# Normalizing and comparing models
############################################################

# Calculate the maximum and minimum confirmed cases to find the range for normalization
max_confirmed <- max(c(covid_train$Confirmed, covid_test$Confirmed))
min_confirmed <- min(c(covid_train$Confirmed, covid_test$Confirmed))
range_confirmed <- max_confirmed - min_confirmed

# Normalize RMSE values for both models and add them to the rmse_results data frame
rmse_results$NRMSE_LightGBM <- rmse_results$LightGBM / range_confirmed
rmse_results$NRMSE_ElasticNet <- rmse_results$ElasticNet / range_confirmed

# Print the updated table of RMSE results with normalized values
print(rmse_results)
```

```
##     Set LightGBM ElasticNet NRMSE_LightGBM NRMSE_ElasticNet
## 1 Set 1 2222.101   3365.274    0.009917834       0.01502012
## 2 Set 2 2114.042   3046.918    0.009435541       0.01359922
## 3 Set 3 2167.036   5200.620    0.009672064       0.02321177
## 4 Set 4 2106.285   3249.034    0.009400918       0.01450131
## 5 Set 5 2151.209   2778.734    0.009601427       0.01240224
## 6 Set 6 1852.237   2752.733    0.008267035       0.01228619
## 7 Set 7 1987.729   4221.327    0.008871769       0.01884092
## 8 Set 8 1876.732   2637.844    0.008376359       0.01177341
```

```
# Determine the best performing model and set
combined_nrmse <- c(rmse_results$NRMSE_LightGBM, rmse_results$NRMSE_ElasticNet)
best_model_index <- which.min(combined_nrmse)
best_model_nrmse <- min(combined_nrmse)
best_model_type <- ifelse(best_model_index <= nrow(rmse_results), "LightGBM", "ElasticNet")
best_model_set <- ifelse(best_model_index <= nrow(rmse_results), best_model_index,
↪  best_model_index - nrow(rmse_results))

# Output the statement on the best model and set
cat("The best performing model is", best_model_type, "using feature Set", best_model_set,
↪  "with a NRMSE of", best_model_nrmse, "\n")
```

```
## The best performing model is LightGBM using feature Set 6 with a NRMSE of 0.008267035
```

The LightGBM model with feature set 6 performed best with a final normalized RMSE of 0.008267035. This is an excellent result as it means the model's predictions are extremely accurate and close to the real numbers. Both models performed well across all feature sets with the highest normalized RMSE still being lower than 0.024.

Conclusion

The objective of this project was to develop a model that could predict future case progression and trends. This is helpful both in the specific case of following COVID-19 data and reacting appropriately, but also in understanding the application of these models to county-level data in the United States.

Two models were used - LightGBM and Elastic Net - and eight feature sets were developed. These included a set with no additional features, three sets with either the lag, rolling average or growth rate feature, three sets with a combination of two additional features, and a set with all three features.

The best model and set combination was LightGBM using feature set 6, which was the set that included the lag and growth rate features but excluded the rolling average features.

The results showed that the inclusion of the rolling average features consistently reduced the effectiveness of the model, increasing the RMSE and normalized RMSE in all models that used the feature. However, the lag and growth rate features were not consistent. In the LightGBM model, the growth rate feature improved the model more than the lag feature. However, in the Elastic Net model it was the reverse. This was consistent across the various sets.

Logically this would make sense as LightGBM is better at capturing nonlinear data than Elastic Net and the growth rates followed a nonlinear trend as seen in the graphs. Meanwhile, the lag feature followed a logistic growth trend that was much less variable.

Future work could add additional features and build much more complex models with more predictors, but that is partly a limitation of the hardware available for this project.

References

[1] D. SKP, "Corona Virus Report," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/imdevskp/corona-virus-report/data. [Accessed: Day-Month-Year].

[2] Microsoft, "LightGBM documentation," LightGBM, 2021. [Online]. Available: https://lightgbm.readthedocs.io/en/stable/. [Accessed: Day-Month-Year].

[3] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," Journal of the Royal Statistical Society: Series B (Statistical Methodology), vol. 67, no. 2, pp. 301–320, Apr. 2005.